分散システムのためのメッセージ表現手法に関する研究

システム情報工学研究科
コンピュータサイエンス専攻
古橋 貞之
指導教員　新城靖

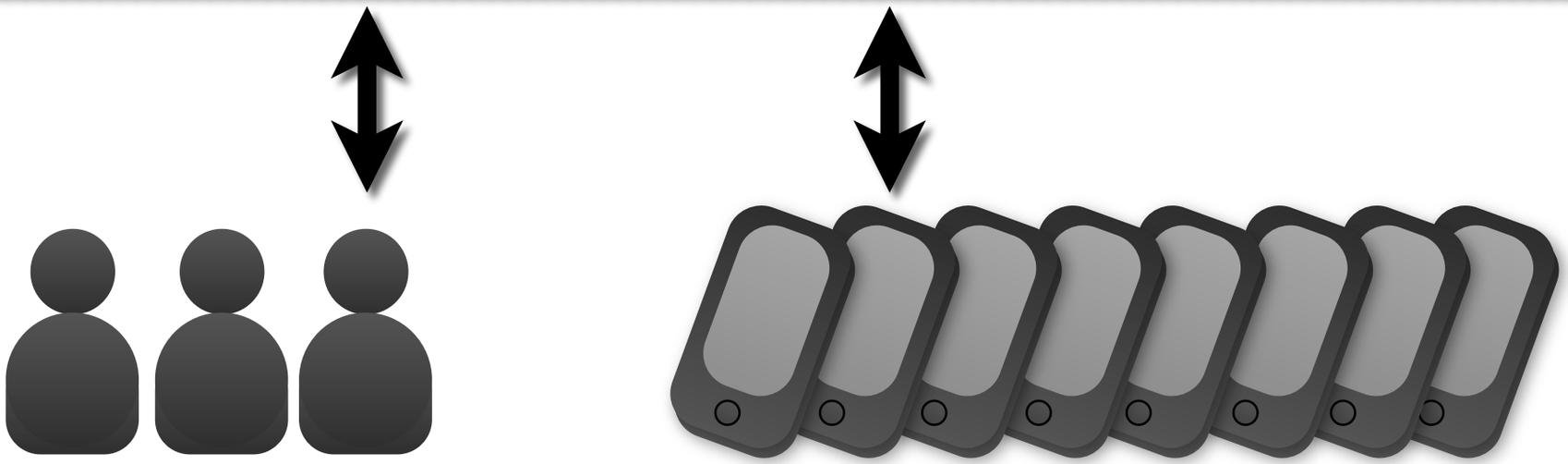
背景

- **ネットワークを通じて不特定多数のユーザーにサービスを提供するシステムが一般化**
  - 複数の機能を組み合わせてシステムを構築
  - 負荷の予測が難しいため、Scale-out型の分散システムが有効

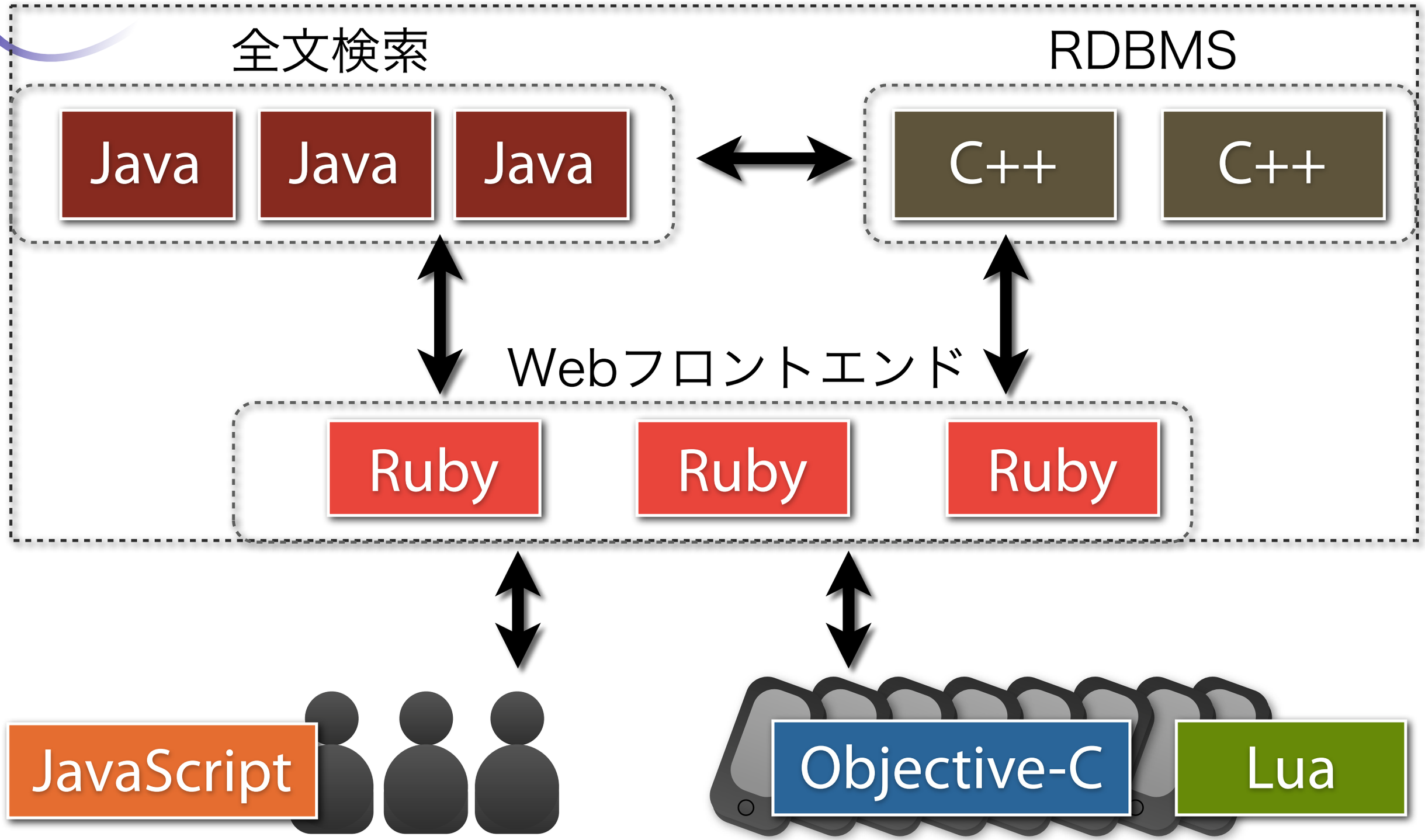
    多数のサーバが相互に通信し、連携することで性能向上を図る
  - ソフト/ハードウェアの更新を繰り返しながら、長期に渡って運用

背景：例

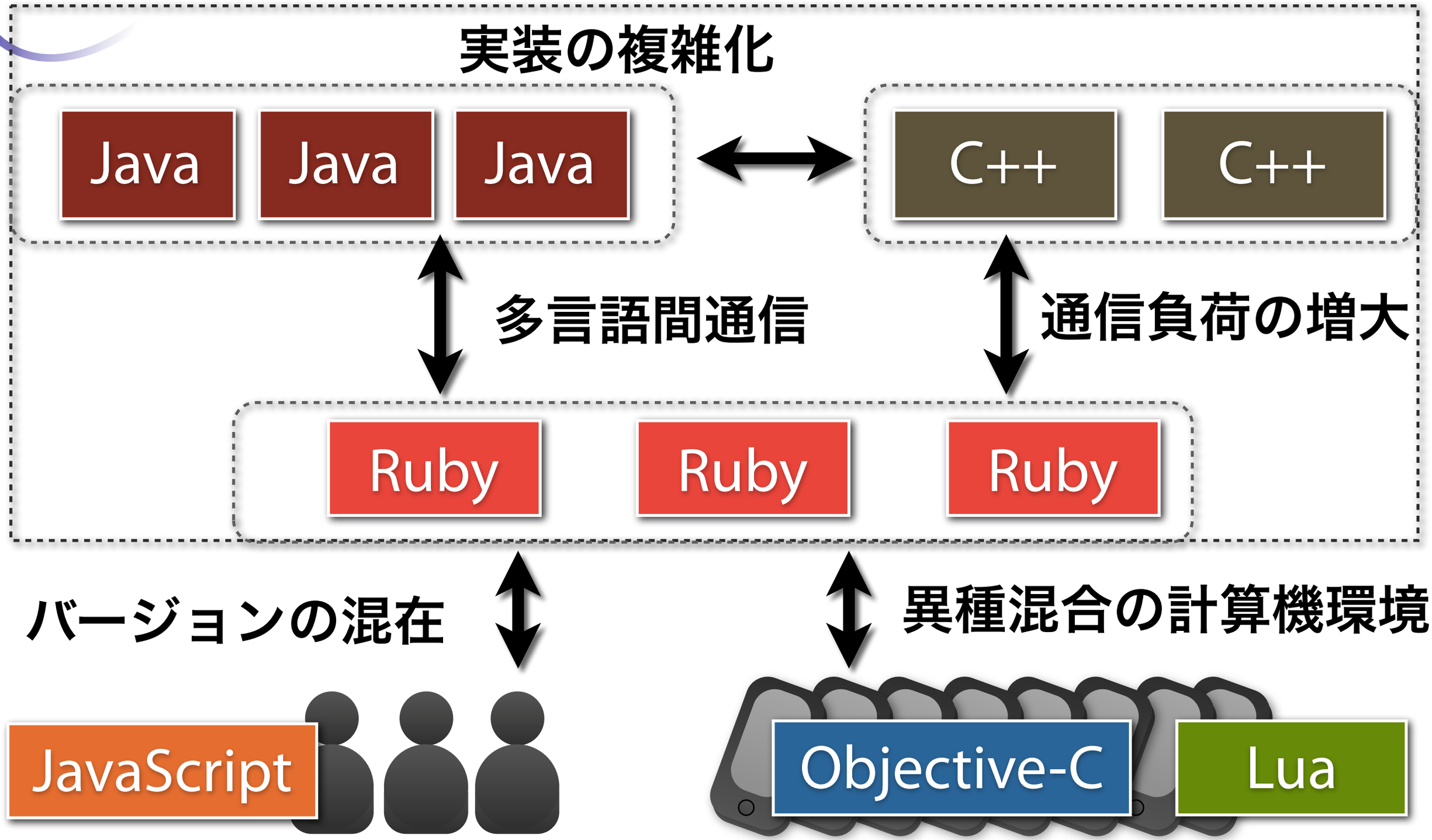
システム



背景：例


全文検索
RDBMS
Java
Java
Java
C++
C++
Webフロントエンド
Ruby
Ruby
Ruby
JavaScript
Objective-C
Lua

背景：分散システムの課題



背景：分散システムの課題

- **実装言語の異なる多種の構成要素が相互に連携**
  - 多言語を横断した通信が必要
- **多数のサーバが相互に通信**
  - 実装の複雑化
  - 通信負荷の増大
- **ソフト/ハードの更新を繰り返しながら長期間運用**
  - バージョンが一致しないソフトウェアが混在
  - 異種混合の計算機環境が混在

背景：求められる機能

- **実装言語の異なる多種の構成要素が相互に連携**
  - 多言語を横断した通信を可能にする
- **多数のサーバが相互に通信**
  - 実装を簡略化する
  - 通信負荷を低減する
- **ソフト/ハードの更新を繰り返しながら長期間運用**
  - バージョン間の互換性を維持した通信を実現
  - 異種混合の計算機環境でも相互運用性を維持

目的

- **新たなメッセージ表現手法を設計・実装**
  - 多言語を横断した通信を可能にする
  - 実装を簡略化する
  - 通信負荷を低減する
  - バージョン間の互換性を維持した通信を実現
  - 異種混合の計算機環境でも相互運用性を維持
- **「MessagePack」**

関連研究

- **JSON (RFC 4506)**
  - 多言語間で利用可能なメッセージ表現手法
  - テキスト形式であるため効率が劣る
  - 静的型検査の仕組みを持たないため、メッセージの正当性を検査する冗長なプログラミングが必要
- **XDR**
  - IDL（インタフェース定義言語）を使用して静的な型への変換を可能にしたメッセージ表現手法
  - 全ノードが最新のIDLを共有している必要がある
  - Cでしか利用できない

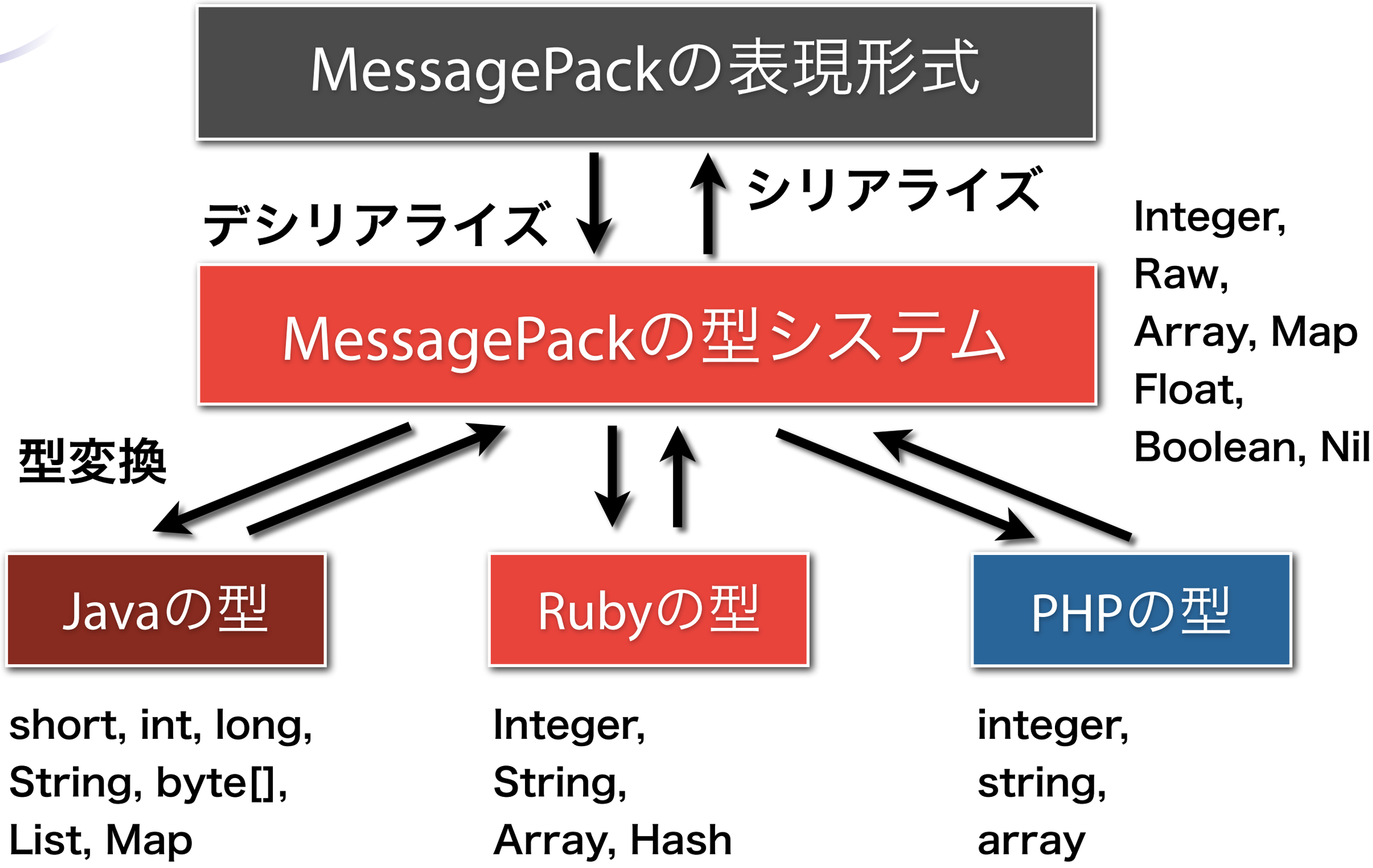
関連研究

- **Protocol Buffers**
  - 多言語間で利用可能なメッセージ表現手法
  - IDLを使用して静的な型への変換が可能
  - 必ず静的な型に変換されるため、型情報を利用したプログラミングができない
- **Avro**
  - IDLをメッセージの先頭に付与することで、IDLの事前共有を不要にしたメッセージ表現手法
  - 各言語の実装でIDL処理系を実装しなければならないため、移植性が低く利用可能な言語が少ない

MessagePackの設計

- **特定の言語に依存しない中立的な型システムを導入**
  - **「MessagePackの型システム」**
  - 各言語の型システムとの相互変換を実装する
  - 多数の言語を横断してメッセージを交換可能に
- **バイナリ列で表現可能にする**
  - **「MessagePackの表現形式」**
  - ネットワークやファイルを経由してメッセージの交換を可能に

設計：型変換モデル


MessagePackの表現形式
デシリアライズ
シリアライズ
MessagePackの型システム
Integer,
Raw,
Array, Map
Float,
Boolean, Nil
型変換
Javaの型
Rubyの型
PHPの型
short, int, long,
String, byte[],
List, Map
Integer,
String,
Array, Hash
integer,
string,
array

実装：言語バインディング

- **プログラミング言語でMessagePackの型システムを扱う方法**
- **「動的型付けオブジェクト」による動的型付け**
  - MessagePackの型システムを直接扱うAPI
- **「型変換テンプレート」による静的型付け**
  - MessagePackの型システムをプログラミング言語の型システムに変換する拡張可能なAPI

動的型付けオブジェクト

- **動的な型付け**
  - 分散システムの実装では、バージョン間の互換性を維持したまま動作し続けるために、実行時に型情報を利用するプログラミング手法が有効
- **MessagePackの型システムを表現するAPIを導入**
  - MessagePackの型システムに基づいた動的型付け
  - 静的型付け言語で実装することで、動的なプログラミング手法を静的型付け言語でも利用可能に

動的型付けオブジェクト

MessagePackの表現形式

デシリアライズ

シリアライズ

型変換せずに直接扱う

MessagePackの型システム

Integer,
Raw,
Array, Map

Javaの型

short, int, long,
String, byte[],
List, Map

Rubyの型

Integer,
String,
Array, Hash

PHPの型

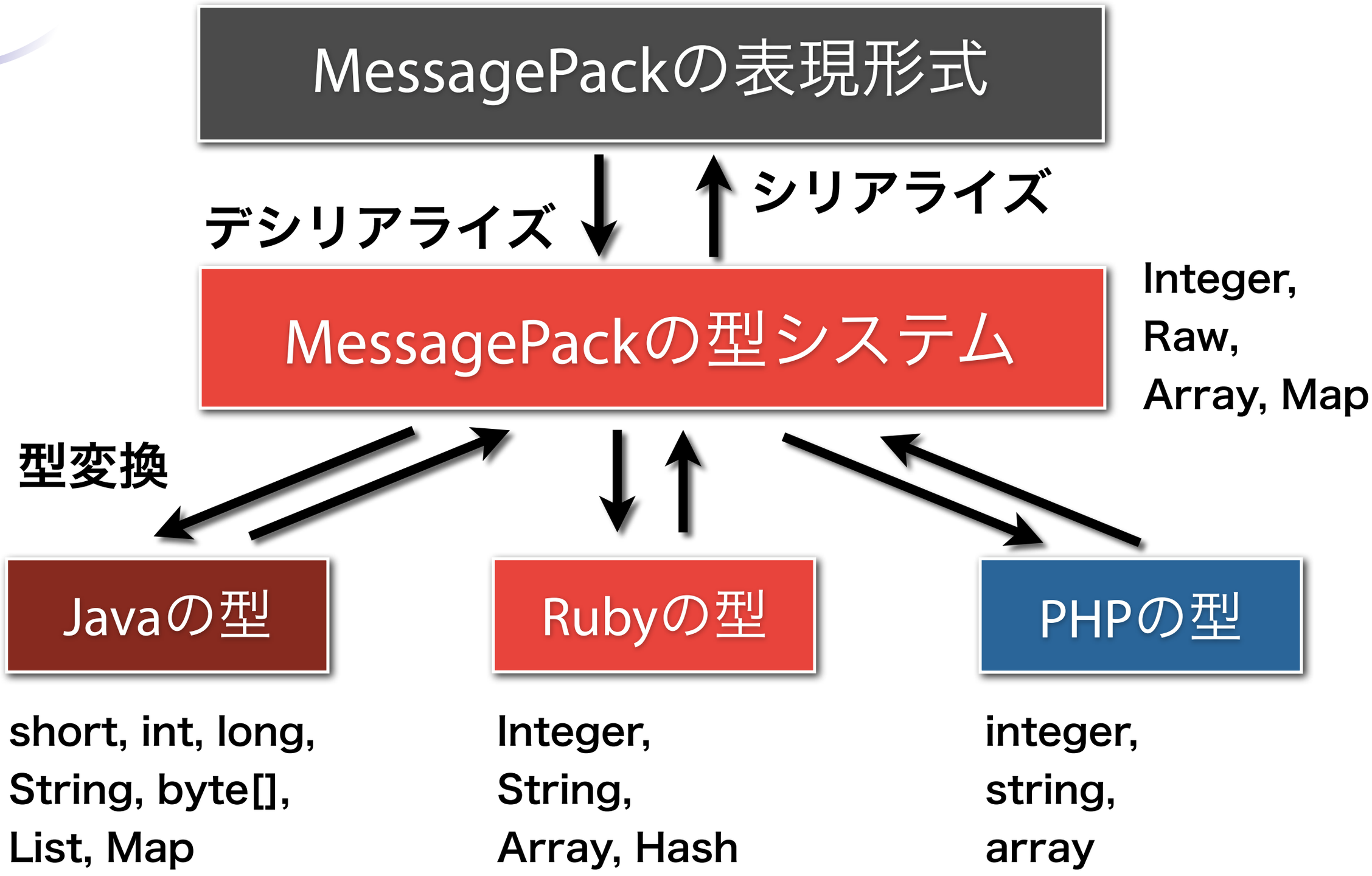
integer,
string,
array

```
public class MyReader {
    public static void main(String[] args) throws Exception {
        // デシリアライズ
        Unpacker unpacker = new MessagePack()
                .createUnpacker(System.in);
        Value obj = unpacker.read();  // 動的型付けオブジェクト

        if(obj.isArrayValue()) {
            // 配列なら、各要素に useValue() を適用
            for(Value v : obj.asArrayValue()) {
                useValue(v);
            }
        } else {
            // そうでなければ、単に useValue() を適用
            useValue(obj);
        }
    }

    public void useValue(Value v) {
        System.out.println("received: "+v);
    }
}
```

型変換テンプレート

- **静的な型検査**
  - 分散システムでは、受け取ったメッセージの正当性の検査が必要（誤動作の防止、デバッグなど）
  - 静的な型検査によって正当性の検査を行うコード量を大幅に削減できる
- **MessagePackの型システムに基づいた動的型付けオブジェクトを静的な型に変換する機構を導入**
  - 静的な型検査を可能にする

型変換テンプレート


MessagePackの表現形式
デシリアライズ
シリアライズ
MessagePackの型システム
Integer,
Raw,
Array, Map
型変換
Javaの型
Rubyの型
PHPの型
short, int, long,
String, byte[],
List, Map
Integer,
String,
Array, Hash
integer,
string,
array

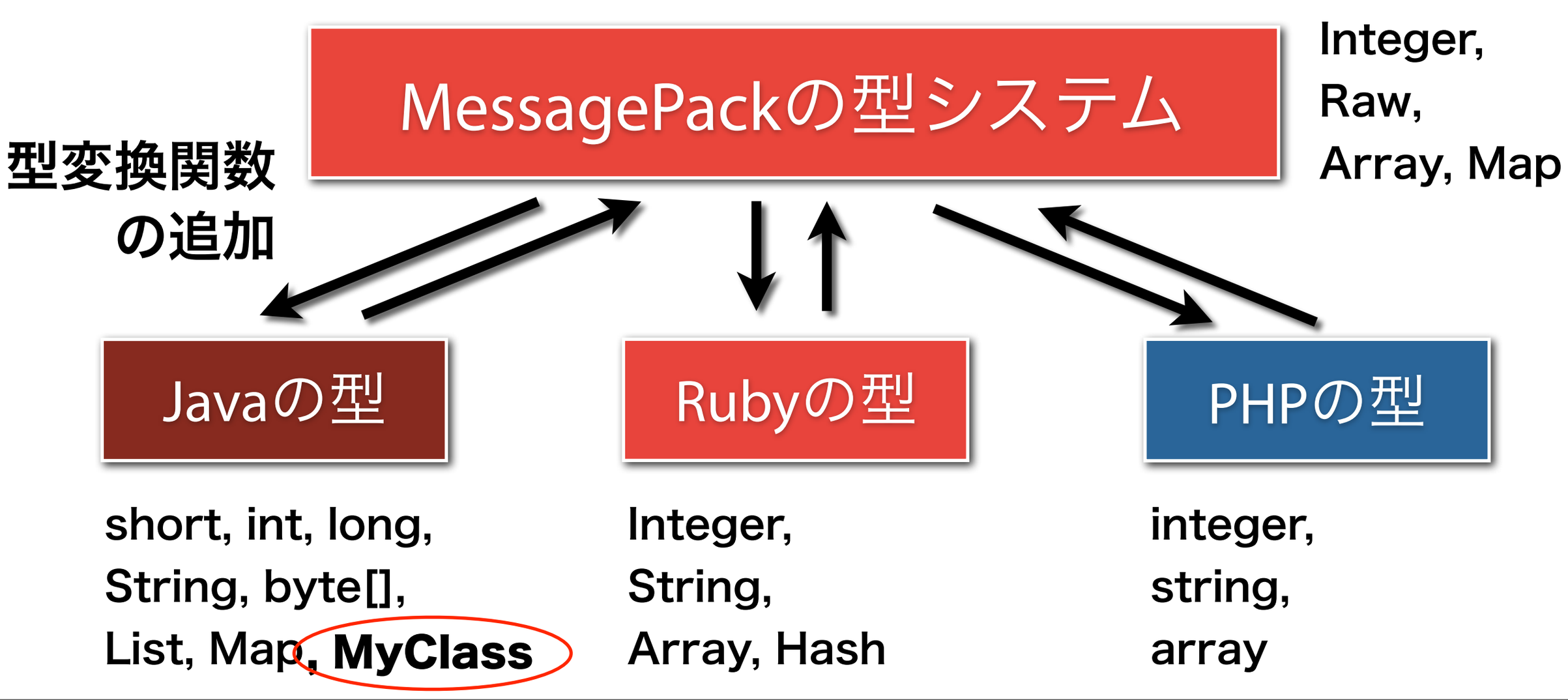
```
public class MyReader {
    static MessagePack msgpack = new MessagePack();

    public static void main(String[] args) throws Exception {
        // デシリアライズを行う
        Unpacker unpacker = msgpack.createUnpacker(System.in);
        Value obj = unpacker.read();

        // 動的型付けオブジェクトを（Javaの）int[]型に変換できるか
        // 検査し、成功した場合は変換して返す
        int[] converted = msgpack.convert(obj, int[].class);
    }
}
```

```
public class MyReader {
    static MessagePack msgpack = new MessagePack();

    public static void main(String[] args) throws Exception {
        // デシリアライズを行う
        Unpacker unpacker = msgpack.createUnpacker(System.in);
        Value obj = unpacker.read();

        // 型検査と型変換...
        if(!obj.isArray() {
            throw new RuntimeException(“invalid type”);
        }

        ArrayValue array = obj.asArray();
        int[] converted = new int[array.size()];

        for(int i=0; i < array.size(); i++) {
            Value e = array.get(i);
            if(!e.isInteger()) {
                throw new RuntimeException(“invalid type”);
            }
            IntegerValue iv = e.asIntegerValue();
            if(iv.getLong() > Integer.MAX_VALUE) {
                throw new RuntimeException(“invalid type”);
            }
            if(iv.getLong() < Integer.MIN_VALUE) {
                throw new RuntimeException(“invalid type”);
            }
            converted[i] = iv.getInt();
        }
    }
}
```

型変換テンプレートの拡張

- **型変換テンプレートのモデル**
  - プログラミング言語の型システムをMessagePackの型システムに射影する機構
  - 型変換テンプレートは射影関数
- **射影関数の追加を可能にする**
  - ユーザーやプログラミング言語の更新によって追加された型に対する型変換を可能にする

型変換テンプレートの拡張



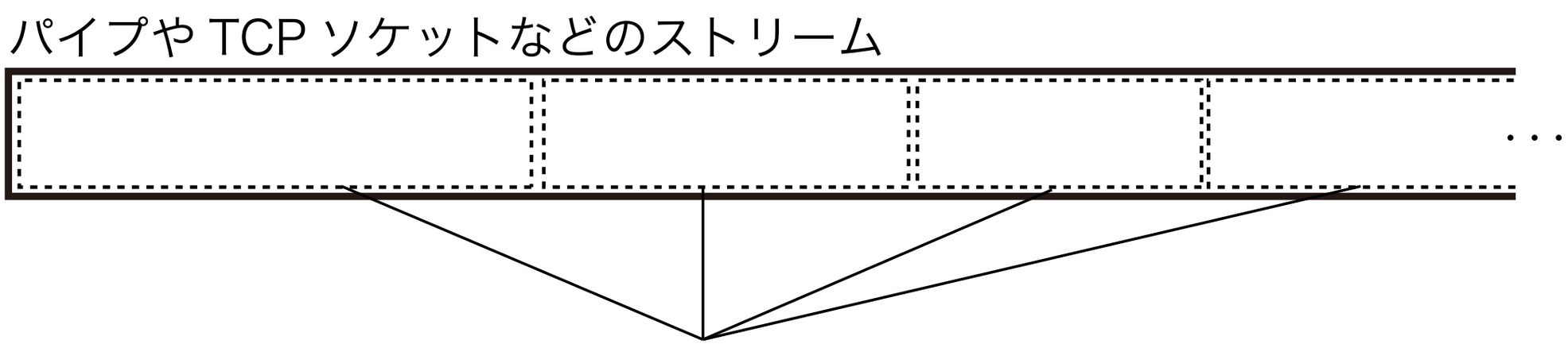
```
public class MyClass {
    static MessagePack msgpack = new MessagePack();

    // 型変換テンプレートを定義
    public static class MyClassTemplate
            extends AbstractTemplate<BigInteger> {
        // ...
    }

    static {
        // 定義した型変換テンプレートを追加
        msgpack.register(MyClass.class, new MyClassTemplate());
    }

    public static void main(String[] args) throws Exception {
        // デシリアライズを行う
        Unpacker unpacker = msgpack.createUnpacker(System.in);
        Value obj = unpacker.read();

        // 動的型付けオブジェクトからMyClassへの型変換が可能になる
        MyClass converted = msgpack.convert(obj, MyClass.class);
    }
}
```

実装の最適化

- **メモリ確保の最適化**
  - スレッドセーフでないが高速なメモリプールを実装（C++）
- **リフレクションの削減**
  - リフレクションを1度だけ呼び出して得られた型情報から生成したプログラムを実行時にコンパイル・ロードすることにより、CPU使用量を削減（Java）
- **コピーの削減**
  - 「ストリームデシリアライザ」の最適化

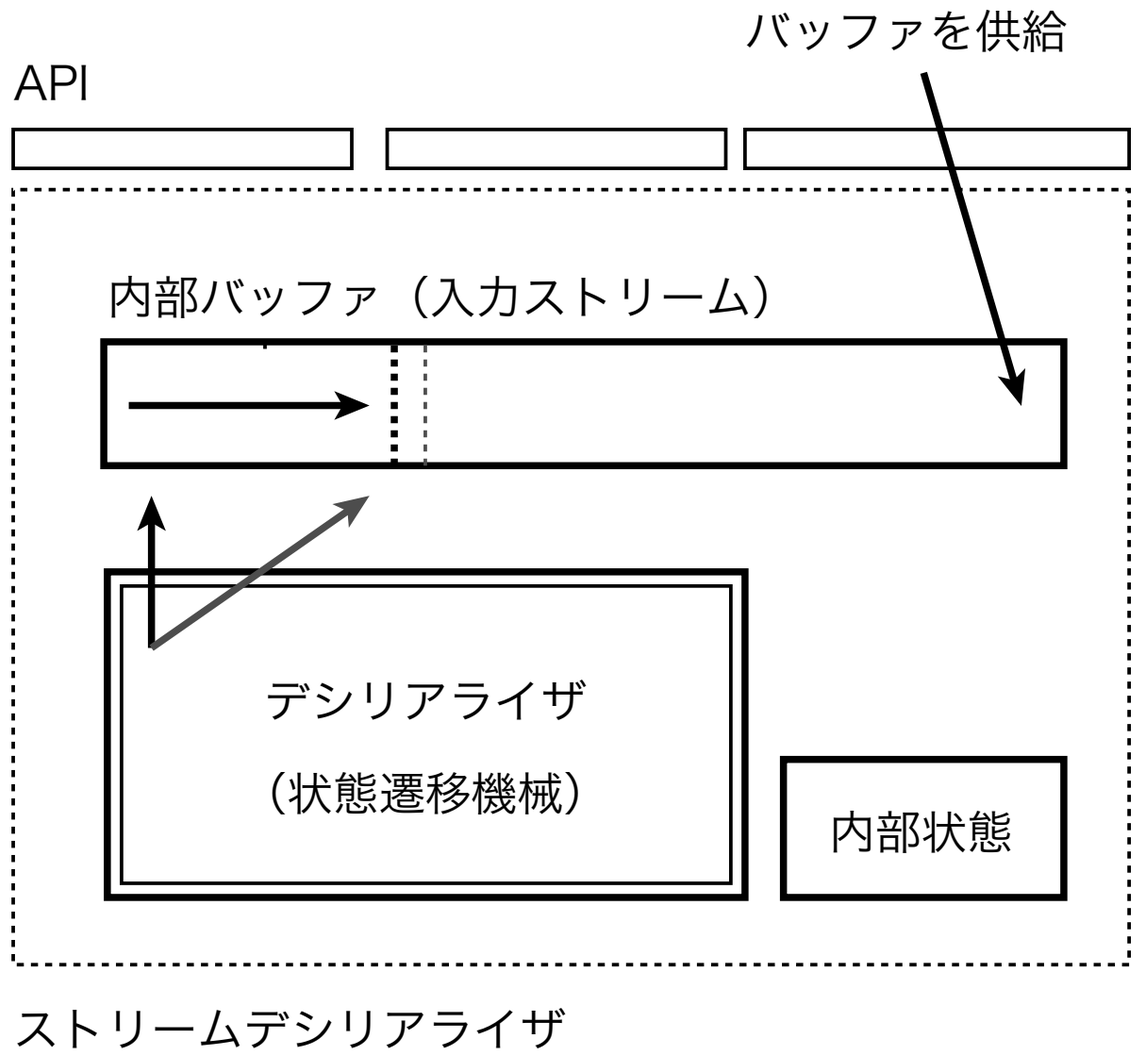
実装：MessagePackストリーム

- **「MessagePackストリーム」**
  - 複数のオブジェクトを次々にシリアライズして繋げたもの
  - 複数のオブジェクトを続けて交換する場合に使用
    TCPソケット上で利用するプロトコル
    ログファイルのファイル構造



実装：ストリームデシリアライザ

- **「ストリームデシリアライザ」**
  - MessagePackストリームを扱うAPI



```
require 'msgpack'

u = MessagePack::Unpacker.new

while buf = STDIN.readpartial(1024)
  # ストリームデシリアライザにデータを供給
  u.feed(buf)  # コピーが発生！

  # デシリアライズを進める
  u.each {|obj|
    puts "deserialized: #{obj}"
  }
end
```

```
require 'msgpack'

u = MessagePack::Unpacker.new

while buf = STDIN.readpartial(1024)
  # ストリームデシリアライザにデータを接続し、
  # デシリアライズを進める
  u.feed_each(buf) {|obj|
    puts "deserialized: #{obj}"
  }
end
```

実装の公開

MessagePackの設計と実装は、
オープンソースソフトウェアとして公開

http://msgpack.org

評価：多言語間通信

```
require 'msgpack'
data = MessagePack.pack([1,2,3])
File.open('data.msg', 'wb') {|f|
  f.write data
}
```

```
public class Main {
    public static void main(String[] args) throws Exception {
        InputStream in = new BufferedInputStream(
                new FileInputStream(new File("data.msg")));
        Value array = new MessagePack().read(in);
        System.out.println("deserialized: "+array);
        //=> deserialized: [1,2,3]
    }
}
```

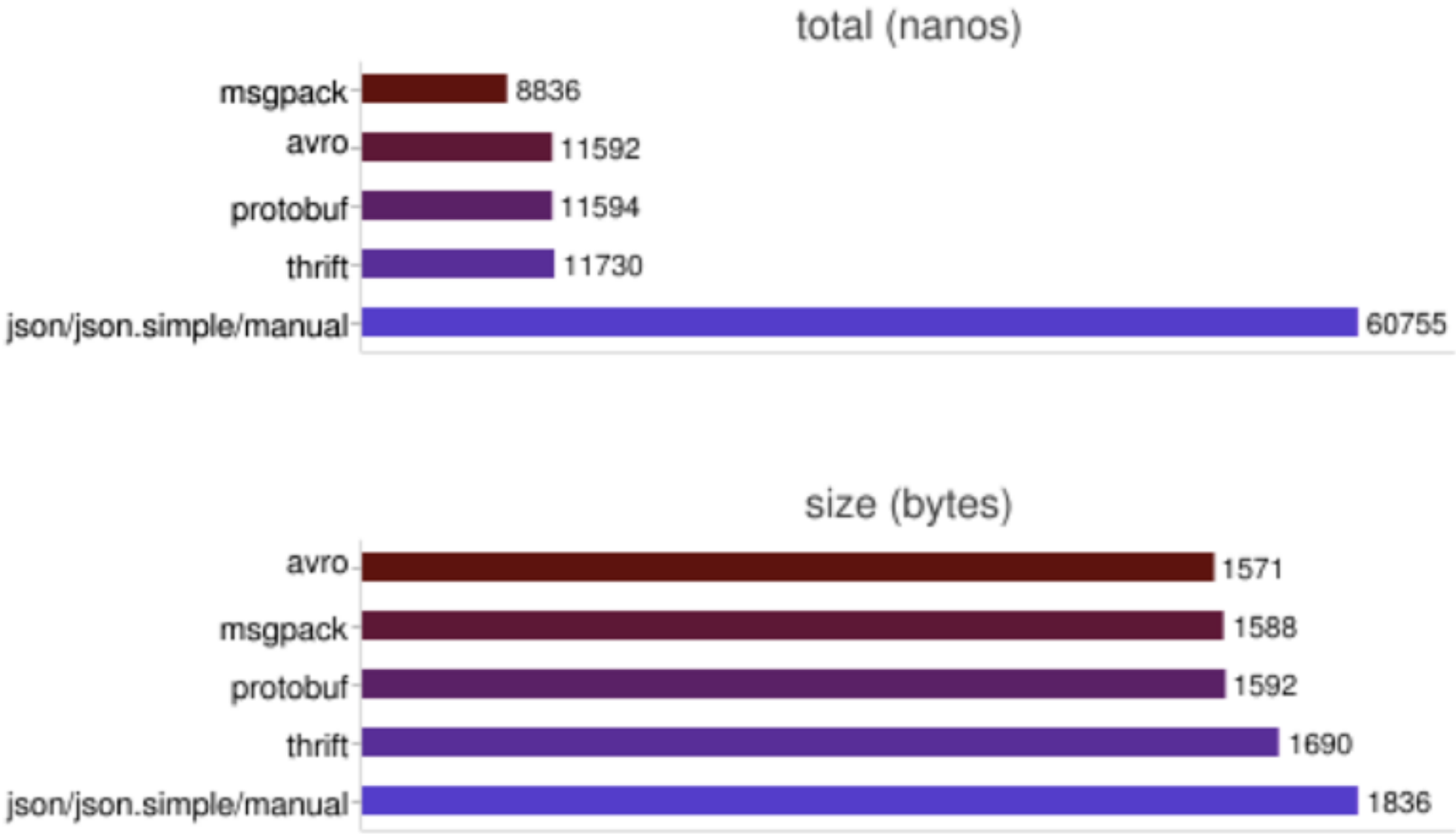
評価：移植性

	シリアライザとデシリアライザ	動的型付け言語	動的型付けオブジェクト	拡張可能な型変換テンプレート
C++	✓		✓	✓
C#	✓			
D	✓		✓	✓
Haskell	✓		✓	✓
Java	✓		✓	✓
OCaml	✓		✓	
Scala	✓		✓	✓
Common Lisp	✓	✓		
Erlang	✓	✓		
JavaScript	✓	✓		
Lua	✓	✓		
Objective-C	✓	✓		
PHP	✓	✓		
Ruby	✓	✓		
Smalltalk	✓	✓		

評価：移植性

- **Protocol Buffers**
  - C++, Java, PHP, Python, Ruby
- **Avro**
  - C, Java, Python
- **MessagePack**
  - C++, Java, PHP, Python, Rubyを含む
    20以上の言語

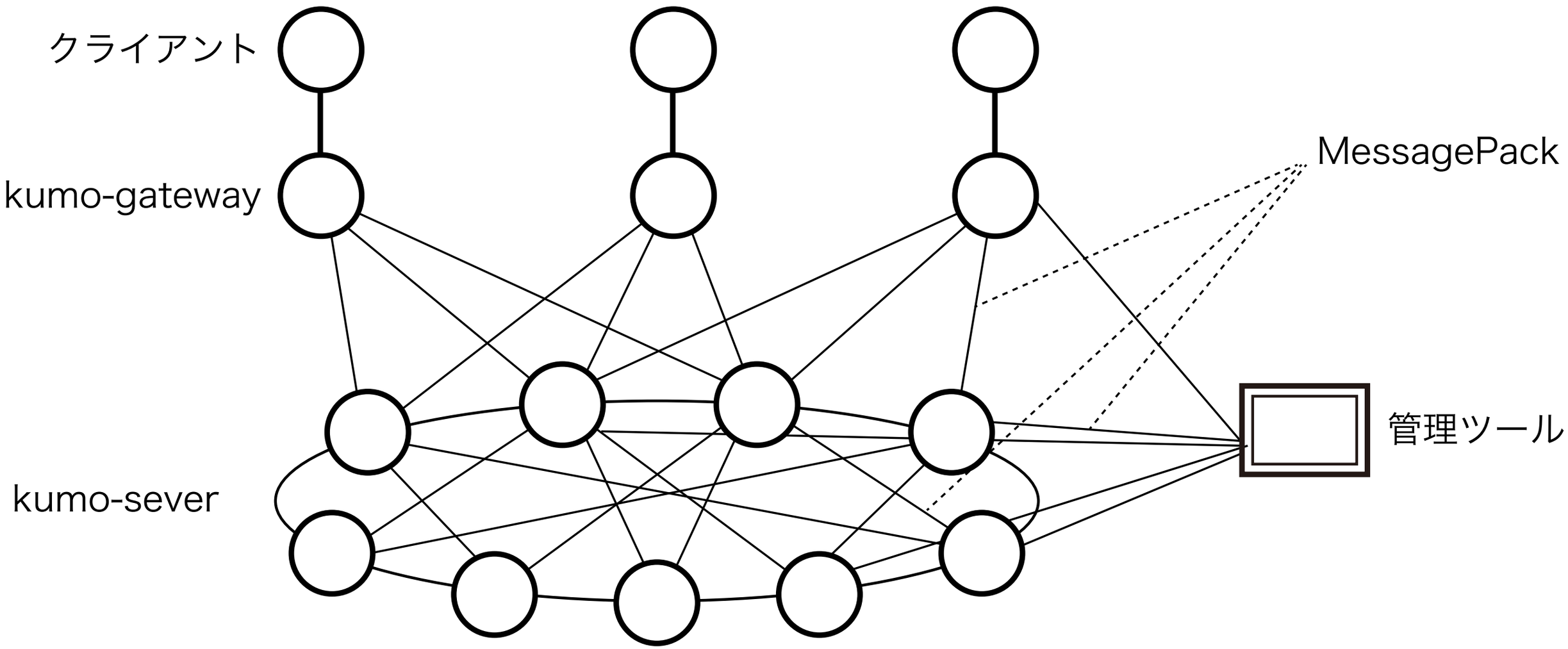
評価：性能評価：Java


total (nanos)
msgpack
8836
avro
11592
protobuf
11594
thrift
11730
json/json.simple/manual
60755
size (bytes)
avro
1571
msgpack
1588
protobuf
1592
thrift
1690
json/json.simple/manual
1836

評価：kumofsの開発

- **高い耐障害性と並列性を重視した分散データストア**
- **サーバ間の通信とサーバ・管理ツール間の通信のプロトコルにMessagePackを使用**
  - 通信のオーバーヘッドを削減
  - 後方互換性を維持したままプロトコルの変更可能であった
  - サーバにはC++を使用して高い並列性を達成
  - 管理ツールにはRubyを使用して高い拡張性を実現
- **実装言語：C++とRuby**
  - 約30,000行

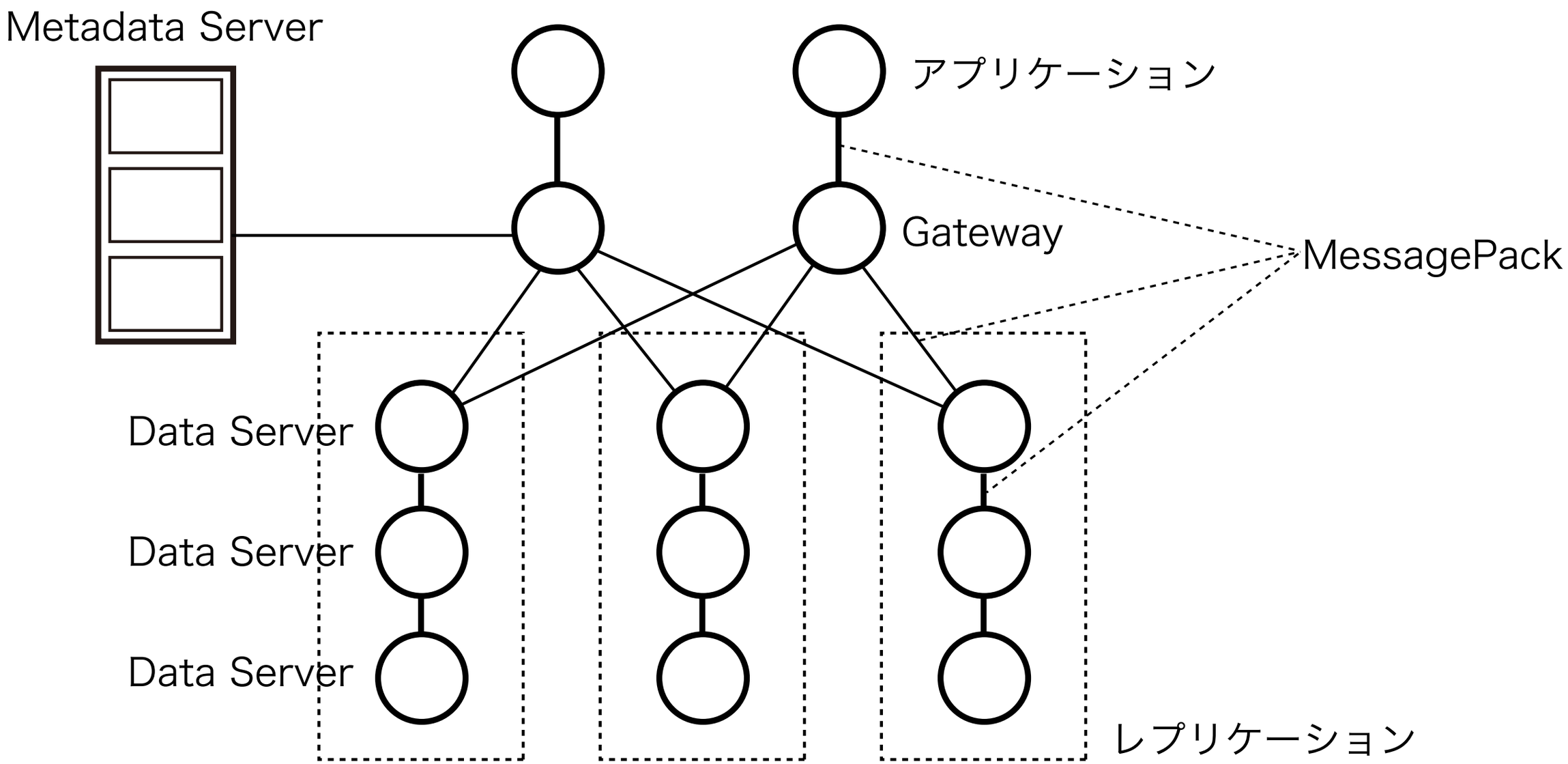
評価：kumofsの開発



評価：LS4の開発

- **高い耐障害性とスループットを重視した分散ストレージシステム**
- **サーバ間の通信プロトコルとレプリケーション・ログの表現にMessagePackを使用**
  - 通信のオーバーヘッドを削減
  - ソフトウェアを改善した際にレプリケーション・ログのデータ構造を変更したが、型を使用したプログラミング手法により後方互換性を維持
- **実装言語：Ruby**
  - 約15,000行

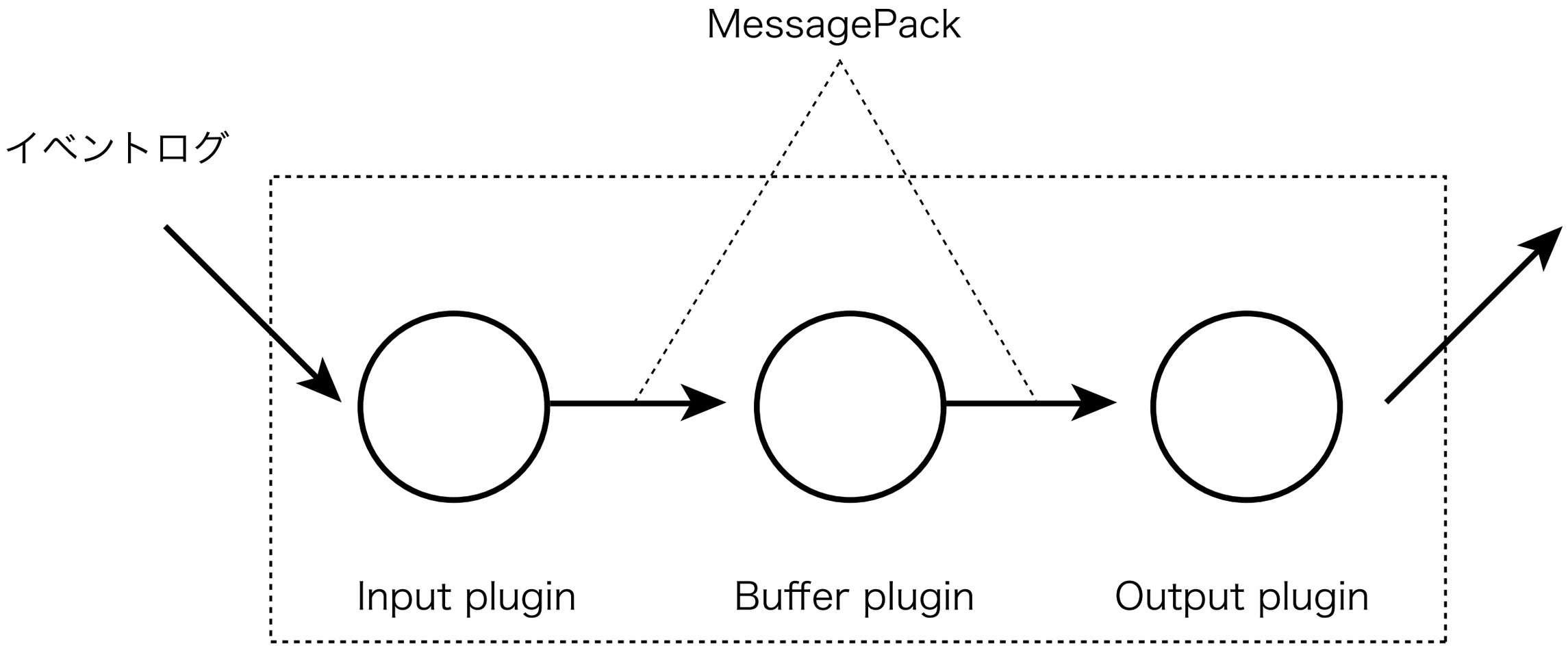
評価：LS4の開発


Metadata Server
アプリケーション
Gateway
MessagePack
Data Server
Data Server
Data Server
レプリケーション

評価：Fluentdの開発

- **プラグイン構造を採用したイベントログ収集ツール**
- **入力・出力・バッファリングの各プラグイン間を組み合わせる事で様々な目的に適合**
- **プラグイン間の通信に使用する内部オブジェクトの型システムにMessagePackを使用**
  - 多言語混合のプラグイン開発を実現
  - 内部オブジェクトをJSONで表現することにより、既存システムとの高い相互運用性を達成
- **実装言語：基本的にはRuby**
  - 約9,000行

評価：Fluentdの開発



まとめ

- **分散システムのための新たなメッセージ表現手法「MessagePack」を設計・実装**
  - 特定の言語に依存しない型システム**「MessagePackの型システム」**を導入
  - MessagePackの型システムのバイナリ列における表現**「MessagePackの表現形式」**を導入
  - MessagePackの型システムと言語の型と相互変換を行う**「型変換テンプレート」**を実装
  - MessagePackの型システムに基づいたオブジェクトを直接扱う**「動的型付けオブジェクト」**を実装

まとめ

- **実装の最適化を行い、高い処理性能を実現**
  - その性能を評価
- **実際的なアプリケーションを開発し、実効性を評価**
  - 異種のサーバ・異種のプログラミング言語・異種のバージョンのプログラムの間で互換性を維持しながら、メッセージを交換することを可能にした
- **今後の課題**
  - 多言語への移植を進めるために、オープンソースソフトウェアとしての開発体制を整備する